GUÍA GRATIS · BYLUNADEV

Los 7 Agujeros

El test de 15 minutos para saber si tu app hecha con IA está perdiendo plata

1 2 3 4 5 6 7

bylunadev

Antes de empezar

Creaste una app con IA. Funciona. La pregunta de esta guía es otra: **¿aguanta que alguien intente romperla?**

Las apps hechas con IA caen siempre en los mismos siete agujeros. La IA optimiza para que *funcione*, y la seguridad es justamente lo que no se ve cuando todo funciona: una app segura y una agujereada son idénticas en pantalla.

Son **siete tests** de unos dos minutos cada uno, para copiar y pegar. Al final sumás los puntos y descubrís tu nota.

Cómo usarla

Hacelos en orden y marcá **Pasó** o **Falló**. Abrí la terminal (en Mac: app **Terminal**; en Windows: **Git Bash** o **WSL**) y pegá estas líneas una vez, cambiando solo la dirección de la app. Guardan la dirección (`$APP`) y el formato de las claves secretas (`$CHAVE`) que usan los tests, así que no tenés que editar nada más. Si cerrás la terminal, pegalas de nuevo.

```
APP=https://tu-app.com
CHAVE='sk-[A-Za-z0-9_-]{20,}|sk_live_[A-Za-z0-9]{20,}'
CHAVE="$CHAVE|rk_live_[A-Za-z0-9]{20,}|gsk_[A-Za-z0-9]{20,}"
CHAVE="$CHAVE|AIza[0-9A-Za-z_-]{35}"
```

Solo en lo tuyo

Corré estos tests **solo en tu propia app**, o en una app cuyo dueño te autorizó por escrito. Probar el sistema de otra persona sin autorización es delito (Argentina: art. 153 bis del Código Penal; Brasil: art. 154-A del Código Penal).

El principio que explica los siete agujeros

> **El navegador es territorio enemigo.**

Todo lo que llega al navegador (código, variables, respuestas) lo puede leer y modificar el usuario. Por eso, los secretos **nunca** van al front, quien decide qué puede hacer cada uno es **el servidor**, y todo dato que viene del navegador es sospechoso hasta que el servidor lo controla. Los siete agujeros son variaciones de esa idea.

AGUJERO 1

La clave de la API está en el front

Qué es. Para que la app hable con la IA (OpenAI, Anthropic, Gemini), usa una clave secreta. Si la llamada se hace directo desde el navegador, la clave viaja adentro del JavaScript que cualquier visitante descarga.

Por qué es grave. Hay robots que barren internet buscando claves expuestas. Cuando encuentran una, usan tu cuenta hasta el límite de la tarjeta. Así nace la factura de miles de dólares.

Test (2 minutos). Primero, desde afuera, como lo vería un robot:

```
curl -sL "$APP" | grep -oE '(src|href)="[^"]+\.js[^"]*"' \
  | cut -d'"' -f2 | sort -u | while read js; do
      case "$js" in http*) echo "$js";; /*) echo "$APP$js";;
                    *) echo "$APP/$js";; esac
    done | xargs -n1 curl -sL | grep -oE "$CHAVE" | sort -u
```

Ese comando solo ve los archivos que carga la página de inicio. Para agarrar **todos** (incluso los que se abren en otras páginas), corré también, dentro de la carpeta del proyecto, después de `npm run build`:

```
grep -rhoE "$CHAVE" dist build out .next/static .output/public \
  2>/dev/null | sort -u
```

¿Sin terminal? Abrí la app en Chrome, **navegá por las páginas** (para que el navegador cargue todo), apretá **F12**, andá a **Sources**, apretá **Ctrl+Shift+F** (en Mac, **Cmd+Option+F**) y buscá `sk-`, `sk_live` y `AIza`.

Resultado	Qué significa
No apareció nada	✅ **Pasó**
Apareció algo parecido a una clave (`sk-…`, `sk_live_…`, `AIza…`)	❌ **Falló**: cambiá la clave **hoy** en el panel del proveedor

Por qué la clave terminó ahí

Las variables con estos prefijos **van al navegador a propósito**: `NEXT_PUBLIC_`, `VITE_`, `EXPO_PUBLIC_`, `REACT_APP_`, `PUBLIC_`. Es el error más común de las apps hechas con IA. Abrí tu `.env` y revisá: ninguna variable con esos prefijos puede guardar un secreto. ¿Usás **Supabase**? La clave `anon` (o *publishable*) puede estar en el front, está hecha para eso. La `service_role` (o *secret*) **nunca**: ignora todas las reglas de acceso de la base.

Clave de Google (Alza...)

Algunas claves `AIza` son públicas por naturaleza, como la de Firebase y la de Google Maps, y aparecen en el front de cualquier app. Solo son un problema si no tienen restricciones. Si la clave que encontraste es la de **Gemini**, es falla. Ante la duda, abrí *Google Cloud → APIs y servicios → Credenciales* y fijate a qué da acceso.

Si falló

Borrar la clave del código no alcanza: ya la copiaron. **Revocá** la clave en el panel del proveedor y generá otra. Después cambiá la arquitectura: el navegador llama a **tu** servidor, y solo el servidor llama a la IA. *App Blindada* muestra el código de ese "proxy" paso a paso.

AGUJERO 2

El `.env` se subió a Git

Qué es. El `.env` guarda las contraseñas de la app: claves de API, base de datos, secreto del login. Si se subió a GitHub, aunque sea una sola vez y después se borró, sigue en el **historial**.

Por qué es grave. Repositorio público: cualquiera lo lee. Privado: cualquier persona que alguna vez tenga acceso (un freelancer, una app conectada) también.

Test (2 minutos). Dentro de la carpeta del proyecto, corré los tres:

1. ¿Algún `.env` se está subiendo ahora?

```
git ls-files | grep -iE '(^|/)\.env' \
  | grep -viE '\.(example|sample|template)$'
```

2. ¿Algún `.env` se subió alguna vez, en cualquier carpeta?

```
git log --all --format= --name-only \
  -- '.env' '.env.*' '*.env' '*.env.*' \
  | grep . | grep -viE '\.(example|sample|template)$' | sort -u
```

3. ¿Quedó alguna clave escrita en el código, en cualquier versión vieja?

```
git log -p --all | grep -oE "$CHAVE" | sort -u
```

Y fijate si el repositorio es público: abrí su dirección en una **ventana de incógnito**. Si abre, es público.

Resultado	Qué significa
Los tres comandos no muestran nada	✅ **Pasó**
Aparece algún archivo `.env` o alguna clave	❌ **Falló**
Repositorio público **y** algún `.env` en el historial	🚨 **Emergencia**: cambiá todas las contraseñas ya

Si falló

Cambiá **todas** las claves del archivo, agregá `.env*` al `.gitignore` y guardá los secretos en las variables secretas del hosting (Vercel, Cloudflare, Railway...). Limpiar el historial solo sirve **después** de cambiar las contraseñas, nunca en lugar de eso.

AGUJERO 3

La IA de la app funciona sin login

Qué es. Tu app tiene una función de IA (un chat, un corrector, un generador). Si la ruta que llama a la IA no controla quién pide, cualquier persona en internet usa tu IA, y pagás vos.

Test (2 minutos).

1. Abrí la app con tu sesión, apretá **F12** y andá a **Network**.
2. Usá la función de IA una vez. Aparece una línea nueva (por ejemplo, `chat` o `generate`). Hacé clic derecho y elegí **Copy → Copy as cURL (bash)**.
3. Pegalo en la terminal, pero **antes de apretar Enter, borrá las líneas que llevan tu identidad**: las que empiezan con `-H 'cookie:`, `-H 'authorization:` o `-b`. Así el pedido llega como si fuera de un desconocido.
4. Al final del comando, agregá `-s -o /dev/null -w '%{http_code}\n'` y apretá Enter.

Resultado	Qué significa
`401`, `403`, o una redirección al login (`302`, `307`)	✅ **Pasó**
`200`	❌ **Falló**: cualquiera usa tu IA gratis
Otro número (`400`, `500`...)	⚠️ No concluyente: fijate si borraste solo las líneas de identidad y repetí

Si falló

La ruta tiene que controlar la sesión **en el servidor** antes de llamar a la IA, y responder `401` si no hay usuario. Esconder el botón en pantalla no protege nada: el ataque no pasa por la pantalla.

AGUJERO 4

No hay límite por usuario

Qué es. Aun con login, un solo usuario (o un robot con una cuenta) puede llamar a la IA mil veces por minuto. Sin límite por usuario, una cuenta gratis es una canilla abierta en tu factura.

Test (2 minutos). Ahora lo contrario del agujero 3: copiá de nuevo la llamada a la IA (**Copy → Copy as cURL (bash)**), esta vez **sin borrar nada**, y pegala dentro de este bucle:

```
for i in $(seq 1 40); do
  PEGAR_ACA -s -o /dev/null -w '%{http_code} '
done; echo
```

Cambiá `PEGAR_ACA` por el comando copiado. Puede tener varias líneas, no pasa nada: lo importante es que el `-s -o /dev/null…` quede justo después de la última.

Resultado	Qué significa
En algún momento aparece `429`	✅ **Pasó**: hay límite
40 veces `200`	❌ **Falló**, salvo que hayas programado un límite mayor a 40 a propósito

Cuidado con este test

Cada llamada que pase es una llamada real y paga a la IA. Cuarenta suelen costar centavos, pero no subas el número.

Si falló

Guardá en la base cuántas veces usó la IA cada usuario **en el día** y rechazá por encima del límite (con `429`). Mejor todavía: un **tope total de la app por día**, para que ni mil cuentas juntas pasen tu presupuesto.

AGUJERO 5

No hay límite de gasto en el proveedor

Qué es. Aunque el código esté bien, un error o un abuso puede gastar más de lo que imaginás. La última línea de defensa es el **límite de gasto en el panel del proveedor de IA**.

Test (2 minutos). Abrí el panel de tu proveedor y buscá:

Proveedor	Dónde buscar
OpenAI	*Settings* → *Billing*: si la cuenta usa créditos prepagos, fijate si la **recarga automática** está activada. En *Settings* → *Limits*, presupuesto y alertas por e-mail
Anthropic	*Console* → *Settings* → *Billing* (recarga automática) y *Settings* → *Limits* (límite de gasto mensual)
Google (Gemini / Cloud)	*Billing* → *Budgets & alerts*. Ojo: el presupuesto de Google **solo avisa, no bloquea**
Cloudflare Workers AI	*Billing*: sin tarjeta cargada, la cuota gratis simplemente se termina; con plan pago, configurá *Notifications*

Resultado	Qué significa
Hay límite **y** alerta, con un monto que podés pagar, y la recarga automática está apagada (o tiene tope)	✅ **Pasó**
No hay, o no sabés dónde está	❌ **Falló**

Si falló

Configurá hoy un límite bajo (por ejemplo, el doble de lo que gastaste el mes pasado) y una alerta al 50%. En una cuenta prepaga, el crédito **es** el límite: dejá poco saldo y apagá la recarga automática. Los paneles cambian de lugar con el tiempo: si no encontrás el menú, buscá *budget*, *usage limit* o *spend limit* en la ayuda del proveedor.

AGUJERO 6

Archivos secretos abiertos en el sitio

Qué es. A veces el `.env`, la carpeta `.git` o un backup terminan publicados junto con el sitio. Hay robots que prueban esas direcciones todo el día.

Test (1 minuto).

```
for p in /.env /.env.local /.env.production /.env.development \
         /.git/config /.git/HEAD /backup.zip /db.sql /dump.sql \
         /.DS_Store /config.json; do
  code=$(curl -s -o /dev/null -w '%{http_code}' "$APP$p")
  printf '%-18s %s\n' "$p" "$code"
done
```

Resultado	Qué significa
Todo `404` (o `403`)	✅ **Pasó**
Algún `200`	❌ **Falló**: abrí la dirección y mirá qué está expuesto

Trampa

Algunos sitios responden `200` a **cualquier** dirección y muestran la página de inicio. Si dio `200`, abrí la dirección en el navegador: si aparece tu home, es falsa alarma; si aparece texto con contraseñas, es real.

Si falló

Sacá el archivo del servidor, **cambiá todo** lo que tenía y ajustá el deploy para publicar solo la carpeta de build (`dist`, `out`, `.next`...), nunca la raíz del proyecto.

AGUJERO 7

Le pediste a la IA que *revise*, no que *ataque*

Qué es. "Revisá la seguridad de mi app" genera una respuesta educada que casi siempre termina en "está todo bien". La IA lee el código con la misma mirada de quien lo escribió. Para encontrar agujeros, tiene que cambiar de lado.

Test (3 minutos para pegar, el resto lo hace la IA). Abrí tu asistente de código (Claude Code, Cursor…) **dentro del proyecto** y pegá:

```
Sos un pentester contratado para entrar a ESTA app, con autorización
del dueño. No revises: ATACÁ. Para cada ruta de la API y cada página
protegida:
1. intentá usarla sin login;
2. intentá ver datos de OTRO usuario cambiando ids;
3. intentá volverte admin o saltear el pago;
4. buscá secretos en el código que va al navegador (incluidas las
   variables NEXT_PUBLIC_, VITE_ y parecidas) y en el historial de git;
5. intentá gastar la IA sin límite;
6. si la app usa Supabase o Firebase, revisá si las reglas de la base
   (RLS / Security Rules) impiden leer y cambiar datos de otro usuario.
Para cada intento, mostrá el comando exacto, el resultado esperado si
está seguro y si creés que es vulnerable. No modifiques ningún archivo.
Al final, listá los hallazgos por gravedad.
```

Resultado	Qué significa
Ya lo hiciste y corregiste lo que apareció	✅ **Pasó**
Nunca lo hiciste, o solo pediste "revisá"	❌ **Falló**

Consejo de quien lo hace en serio

La IA también se equivoca cuando ataca. Puede decir que algo es vulnerable cuando no lo es, y al revés. **Confirmá cada hallazgo corriendo el comando vos misma o vos mismo** antes de creerle.

Tu nota

Sumá **1 punto por cada test que pasó**.

Puntos	Diagnóstico
7	🟢 **Base sólida.** Cerraste los agujeros que tiran abajo a la mayoría de las apps. El siguiente nivel es autorización (un usuario viendo datos de otro), panel de admin, fuerza bruta en el login y dependencias.
5–6	🟡 **Expuesta.** Uno o dos agujeros te pueden salir caros. Corregí los que fallaron esta semana, empezando por el 1, el 2 y el 3.
3–4	🟠 **Vulnerable.** Es cuestión de tiempo hasta que un robot la encuentre. Pará de sumar funciones y corregí antes.
0–2	🔴 **Emergencia.** Cambiá las claves hoy y poné un límite de gasto ya, antes que cualquier otra cosa.

Pasar los 7 no es estar segura

Estos tests detectan los agujeros más **comunes y caros**. No cubren todo. Una app puede pasar los siete y aun así dejar que un usuario vea los datos de otro cambiando un número en la URL, que es el agujero más común *después* de estos.

El próximo paso

Esta guía te mostró **dónde** están los agujeros. Cerrar cada uno, y encontrar los que no entran en un test de dos minutos, es el tema de **App Blindada**.

App Blindada · Guía técnica de ciberseguridad para quienes crean apps con IA

- El código de la corrección de cada uno de los 7 agujeros, y de los que vienen después.
- Autenticación y autorización: fuerza bruta, sesión, panel de admin y el agujero de "cambiar el id en la URL".
- Tope de gasto de la IA en el código, a prueba de dos pedidos al mismo tiempo.
- Inyección, XSS, CORS, webhooks y headers de seguridad.
- El guion completo para que la IA ataque tu app, con prompts por área.
- El caso real de una plataforma auditada, con cada ataque y su resultado.
- Un checklist de 47 ítems con **Nota de Seguridad** de 0 a 100.

Y si preferís que lo haga alguien por vos: **hago la auditoría de tu app**, con contrato, ataque autorizado, informe y retest después de las correcciones.

> **Hablame:** @bylunadev